NPSCS-92-016

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Managing Real-Time Software Projects: Problems and Issues

Luqi, M. Shing, V. Berzins & L. Chmura

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

FedDocs
D 208.14/2
NPS-CS-92-016

DOE 14121
NPS-03-94-016 c 2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Rear Admiral R.W. West, Jr.
Superintendent

HARRISON SHULL
Provost

This report was prepared with research funded by the Naval Research Funds provided by the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:

UNCLASSIFIED
ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
NPSCS-92-016

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Computer Science Dept.
Naval Postgraduate School

6b. OFFICE SYMBOL *(if applicable)*
CS

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA 93943

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Naval Postgraduate School

8b. OFFICE SYMBOL *(if applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA 93943

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Managing Real-Time Software Projects: Problems and Issues

12. PERSONAL AUTHOR(S)
Luqi, M. Shing, V. Berzins & L. Chmura

13a. TYPE OF REPORT	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
		1992, December, 31	13

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Hard real-time systems are defined as those software systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced. Hard real-time systems of interest to DoD are tailored specifically to their applications, employ various degrees of fault-tolerance, and are typically embedded in larger systems. The process of design and development of these systems is often plagued with uncertainty, ambiguity and inconsistency. This report examines problems and issues in managing real-time software projects.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Luqi

22b. TELEPHONE *(Include Area Code)*
(408) 656-2735

22c. OFFICE SYMBOL
CSLq

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

MANAGING REAL-TIME SOFTWARE PROJECTS: PROBLEMS AND ISSUES

Luqi, M. Shing and V. Berzins
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Lou Chmura
Naval Research Laboratory
Washington, D.C. 20375

1. INTRODUCTION

"On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Dessert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans.

"The Patriot battery at Dhahran failed to track and intercept the Scud missile because of a software problem in the system's weapon control computer. The problem led to an inaccurate tracking calculation that became worse the longer the system operated. At the time of the incident, the batteries had been operating continuously for over 100 hours. By then, the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud.

"The Patriot had never before been used to defend Scud missiles nor was it expected to operate continuously for long periods of time." [1]

The single most costly event in the Operation Dessert Storm was the result of a software timing error. It underscores the importance and difficulties in the design and development of hard real-time software in our military systems. This report examines problems and issues in managing real-time software projects.

2. CHARACTERISTICS OF HARD REAL-TIME SOFTWARE SYSTEMS

Hard real-time systems are defined as those software systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [2]. Hard real-time systems of interest to DoD are tailored specifically to their applications, employ various degrees of fault-tolerance, and are typically embedded in larger systems. Major characteristics of hard real-time software systems include:

a) Complex and uncertain environment

Hard real-time software systems are typically embedded in larger systems, performing critical control functions. These real-time control functions require the software system to interact with a wide variety of hardware/software subsystems via networks. Because these hardware and software systems undergo steady evolution and their precise specifications are often lacking, it is very

difficult to establish the initial requirements of the real-time systems to capture the problems correctly. It is also difficult to determine the impact of each change to the system's operating environment on software requirements, designs, and implementations as well as on the project cost and schedule.

b)Stringent Software Reliability Requirements

Severe consequences may result if timing as well as logical correctness properties of the system are not satisfied. Hence, all hard real-time systems require a high level of system reliability. A *failure* in a real-time system can be defined as a deviation of the behavior of the system from its specified behavior. Failures are caused by either mechanical (hardware) or algorithmic (software) errors called *faults*. Two concepts are involved in building reliable systems: *fault-prevention* and *fault-tolerance*. Fault-prevention is concerned with eliminating faults from the system before it goes on line. There are two stages to fault-prevention: *fault avoidance* and *fault removal*. Fault avoidance attempts to limit the introduction of potentially faulty components during the design and implementation of the system. However, despite the best effort of avoidance techniques, faults will inevitably be present in the system. It is then the concern of fault removal to find and remove the causes of these errors. Fault-tolerance, on the other hand, is concerned with continuing acceptable operation even if a failure in a subsystem does occur. Note that fault-tolerance facilities must be included in all hard real-time systems even if the systems are fault-free. The reason for this is that "fault-free" can only mean that the system corresponds to its specification. The specification may contain errors or omissions and may be based on incorrect assumptions about the system's environment. Real-time systems are often required to function correctly despite the occurrence of some hardware and/or software failures, and/or human errors.

c) Typical Designs have Unpredictable Behavior

Typical tactical control software consists of numerous processes including communication processing, known-object recognition, track processing, and display processing among others. While conceptually these processes could frequently run concurrently, limitations in hardware resources often force these processes to be sequentialized in a centralized implementation through an interrupt driven prioritization scheme. This scheme produces a completely dynamic schedule whose effects are difficult to predict, control, and certify.

3. MAJOR ISSUES IN MANAGING REAL-TIME SOFTWARE PROJECTS

The process of design and development of hard real-time software systems is often plagued with uncertainty, ambiguity and inconsistency. The timing requirements are difficult for the user to provide and for the analysts to determine. It is also very difficult to determine whether a delivered system meets its requirements. In this paper, we focus our attention on four major issues in managing real-time software projects: software quality, productivity, maintenance, and systems acquisition/integration.

a) Software Quality

i) Complexity demands accuracy

As the real-time embedded systems become increasingly complex, human understanding becomes a limiting factor for building large reliable systems. No single person can fully understand the whole system. Since the probability of faults increases exponentially with the size of the

system and the development team, we have to impose very stringent requirements on the accuracy of the software development process. The stringency of these accuracy requirements gets so severe for large systems that qualitatively new technologies are needed to fulfill them, such as software tools capable of preventing or locating errors in a system design and implementation.

Every project manager must therefore face the question "*Are we building the product right?*". *Verification* involves checking that the system conforms to its specification [3]. Due to the potential severe consequences resulting from system failures, it is vital to have the correct specification for the hard real-time systems and to build the systems to fully meet the correct specifications. Unfortunately, hard real-time software systems are typically very large, and large systems are more error prone. A precise and formal specification is essential to system verification. The major problem with formal specification and verification is that the field is still very new. There is no consensus (not to mention standards) on the right model and right method. Almost all of the existing formal specification and analysis methods are hard to use due to the lack of user-friendly tools and integrated environments, and there is a trade-off between ease of use and sufficient richness of the underlying models to express all of the issues that arise in real projects. There has been progress in the formal specification and analysis of real-time systems. These formal techniques have influenced the design of languages which are needed for programming-in-the-large and software tools for software engineering. A number of formal languages for specifying and reasoning about the requirements of real-time software systems have become available in recent years. These include tabular methods, graphical methods, logic-based languages, and several process algebras. Recent success stories in formal approaches to system development include: (i) formal specification and verification has been used successfully in hardware development projects [7] and VHDL specifications have been required for government hardware development projects, (ii) formal security models and formal verifications techniques are being used by NSA to certify the security of the operating systems and (iii) NRL has successfully demonstrated the usefulness of Modechart specification and the SARTOR verifier [8, 9]. Unfortunately, these success stories only amount to tiny drops in a big ocean. Technology transfer is a very slow and difficult process. Continued commitment from the management to modernize the software development methodology and to support research and development of better tools and environments for the formal methods is needed to create the capability to routinely develop reliable hard real-time systems.

ii) Communication requires different forms of specification

One major question faced by every project manager is "*Are we building the right product?*". To build the right system, one must first understand what the customer really wants. The process of establishing the services which the target system should provide and the constraints under which the system must operate is called *requirements analysis*, and the process of checking that the target system, if implemented according to the specification, meets the expectations of the customers is called *validation* [3]. The result of this analysis and validation is a requirements specification which often constitutes the first formal document produced in the software process.

Real-time embedded systems are typically application specific. Most customers have only sketchy ideas on how the target system should behave and software developers often lack complete knowledge about the operating environment surrounding the system to be developed. Hence, it is very important to express the requirements for a software system at different levels of abstraction,

with different degrees of formality, and to help the customer visualize the behavior of the target system satisfying the requirements. The highest level requirements are usually informal and imprecise, but they are understood best by the customers. The lower levels are more technical, precise, and better suited for the needs of the system analysts and designers, but they are further removed from the user's experiences and less well understood by the customers. (Interested readers can refer to the NRL report by Clements *et al* for a set of evaluation criteria for real-time specification languages [4].) Because of the differences in the kinds of descriptions needed by the customers and developers, it is not likely that any single representation for requirements can be the "best" one for supporting the entire software development process [5]. One way to bridge the communication gap between the customers and developers is by means of computer-aided rapid prototyping [6]. The desired computer-aided rapid prototyping environment should provide user-friendly tools for the customers/analysts to walk through the high-level design and define the initial requirements as graphical objects, tables, and high-level semi-formal specifications. These high-level requirements are then translated by the prototyping tools into low-level specifications which are formal, precise and unambiguous, meeting the needs of the system analysts and designers. The computer-aided rapid prototyping environment should also provide tools to help the designer construct executable prototypes based on these low-level formal specifications. The demonstrated behavior of the executable prototype, in turn, provides concrete information for the customer to validate the requirements and to refine them if necessary.

iii) Portability requires Language Support

All approaches to programming real-time systems share a common characteristic: to control the execution time so that the program will meet its timing constraints even under worst case conditions. Ada83 was originally designed for embedded system implementation and does improve on previous languages in some respects, but the design of real-time systems was not well understood in 1983. Consequently, Ada83 it is not ideal for implementing hard real-time systems. The designers of Ada83 have made a number of fundamental errors from the perspective of current technology. Ada83 does not provide attributes to explicitly express task deadlines or timing constraints that define the start, finish and maximum duration allowed for a constraint block. These problems are intrinsic to Ada83 and cannot be resolved by any COTS real-time operating systems. The Ada9x committee [14] is considering additional timing features, so we are likely to see at least some of these features in future versions. Currently, many hard real-time systems are still written in assembly language so that tight deadlines can be met. However, assembly codes are very costly to build, maintain, and very hard to test. Commitment to developing and adopting a standard such as Ada9X is needed to achieve portable hard real-time software that can be developed and maintained at reasonable cost.

b) Productivity

i) Large Development Teams need Tool Support for Coordination

Managers for real-time software projects must address problems common to developing any large software system. As systems get larger, larger teams of engineers are needed to build the entire system. We need a methodology that can integrate systematic planning and management with formal methods and design automation [10], and tools that can help coordinate the efforts of large teams [17].

ii) Quick Development requires Automatic Generation of Schedules

Manual approaches to creating real-time schedules are very labor-intensive and are not very reliable, since cut-and-try methods rely on testing to detect timing faults. There is no systematic way to tell when the process is really done, because absence of timing faults on any given set of test data does not guarantee that a different set of test data will not expose more faults. Automatic methods for generating schedules that are guaranteed to meet all the timing constraints if the method terminates successfully can greatly speed up the process and make it more predictable.

iii) Personnel Turnover requires Computer-Aided Design Management

As the number of people in the team increases, the fraction of each worker's time devoted to communication increases while the fraction of time left for software development decreases. To cut down the communication overhead, we need precise design documents that can be managed on-line. The need for precise documentation is intensified in real-time system development because requirements of the sub-modules are changed frequently during over the course of development in order to meet the tight timing constraints. Furthermore, many people may enter and leave the development team in the middle of a large project. When a member of the original design team leaves the project, undocumented critical information that is lost can be very costly or impossible to recover. Hence, we need a methodology that supports automated design management in addition to on-line precise design documentation.

iv) Software Reuse needs Tool Support

One way to improve productivity is through code reuse. However, like formal specification of real-time systems, software reuse technology is still at its infancy stage. There is a lack of standards in the specification of the reusable components, and lack of effective automated tools for finding relevant components, assessing their suitability, and adapting them to particular applications. Better tools of these kinds are needed to increase the practical impact of software reuse.

c) Maintenance

i) Flexibility requires Scheduling Support

Evolution of real-time systems is particularly difficult because real-time constraints and finite computing resources introduce dependencies between all of the functions of the system, most of which would otherwise be independent. If such systems are implemented manually, then even small changes can result in major redesign efforts because the schedule controlling the execution of the time-critical processes must be reworked, and the boundaries of many seemingly unrelated sub-functions may have to be readjusted throughout the entire system. We need tools to automate the scheduling process and take the guesswork out of the schedule adjustments. Such tools will enable developers to adapt systems to changing requirements more quickly and with greater reliability.

ii) Re-engineering requires Modifications to Requirements and Designs

One common misconception about re-engineering is that one can write a formal specification from an existing design or an existing program, and then use the formal specification for consistency/completeness checking and code re-development. The major problem with this approach is that developing formal specifications from existing designs inevitably exposes errors in the specifications and designs. This requires changing the design as part of the process. In one project that was constrained to specify the system as implemented, errors and all, there wound up being two versions of the specifications, the current one and the correct one. This led to a subsequent effort

to change the code to conform to the correct specification.

d) Systems Integration and Acquisition

i) Early System Testing and Integration requires Prototyping

Traditional software development methods conduct extensive testing near the end of the project in an attempt to ensure proper functioning of the system. The major weakness of this approach is that there is no way to recover from major faults discovered at the end of the project, when available funds and time have been nearly exhausted. Furthermore, delivered systems that meet faulty requirements can satisfy a contract without being useful to the customer. One way to overcome the above weakness is to use a prototyping method that spans the entire life-cycle of the real-time software [11, 12]. Unlike throw-away prototypes, the prototyping method should provide requirements and designs in a form that can be used for comparison during system testing. The existence of a flexible prototype can significantly ease system testing and integration. When final implementations of subsystems are delivered, integration and testing can begin before all of the subsystems are complete by combining the final versions of the completed subsystems with prototype versions of the parts that are still being developed.

ii) Prototyping Technology can help to Evaluate Delivered Systems

Two things are needed for a successful acquisition: to make sure that the system is delivered on time and that the delivered system meets its requirements. A computer aided rapid prototyping environment, besides being a useful tool to the hard real-time system developers, is also very useful to the customers [13]. Acquisition managers can use the system to ensure that acquisition efforts stay on track and that contractors deliver what they promise. The prototyping system enables validation of requirements via prototyping demonstration, greatly reducing the risk of contracting for construction of systems that do not meet customer needs. The validated high-level designs resulting from prototyping can be used to formulate subcontracts for subsystems that prevent many system integration problems. Other problems can be detected earlier by testing interactions between delivered subsystems and prototypes of subsystems not yet complete. Prototypes generated by the prototyping system should include structures for experimentally measuring whether critical components meet their timing requirements. These structures can be used to evaluate real-time performance of systems delivered by contractors. The prototyping system should provide automated functional testing that can compare results of delivered systems to results produced by the prototype.

4. RECOMMENDATIONS

a) Hard real-time software system developers

In addition to all the knowledge needed for developing large software systems, hard real-time software system developers should also be familiar with methods for modelling complex hard real-time systems and scheduling hard real-time tasks. An introduction to these two subjects can be found in [2].

b) Hard real-time software system environment

Computer-aided tools is vital to the success of developing reliable hard real-time software. There are many COTS software tools which support the design and coding of soft real-time software

systems (see [15, 16] for details). However, there are few COTS tools available to support the development of hard real-time systems, and we have seen none that can take requirements all the way to generated schedules and generated production-quality code. The major difference between soft real-time systems and hard real-time systems is the existence of deadlines and the requirement of meeting these deadlines under worst-case situation in hard real-time systems. Two things are needed to support the development of hard real-time systems:

(i) requirement and programming language support for specifying timing constraints such as maximum communication delays between processes, the maximum time allowed from the triggering of the process to the completion of the process, the frequency of periodic operations, etc.

(ii) automatically generated schedules that guarantee meeting all deadlines of timing-critical processes under the worst-case situation.

With the exception of CAPS [13], none of the existing COTS case tools generate code that guarantees meeting all hard real-time requirements.

5. REFERENCES

[1] GAO Report, "Patriot Missile Defense - Software Problem Led to System Failure at Dhahran, Saudi Arabia", GAO/IMTEC-92-26, United States General Accounting Office, Washington, D.C. 20548, Feb. 1992.

[2] J.K. Stankovic and K. Ramamritham, *Tutorial on Hard Real-Time Systems*, IEEE Computer Society Press, Washington, D.C., 1988.

[3] B.W. Boehm, "Software engineering: R&D Trends and Defense Needs", in *Research Directions in Software Technology* (P. Wegner ed.), MIT Press, Cambridge, MA, 1979.

[4] P.C. Clements, C.E. Gasarch and R.D. Jeffords, "Evaluation Criteria for Real-Time Specification Languages", *NRL Memorandum Report 6935*, Naval Research Laboratory, Washington, D.C., Feb. 1992.

[5] Luqi, R. Steigerwald, G. Hughes, F. Naveda, and V. Berzins, "CAPS as Requirements Engineering Tool", *Proc. Requirements Engineering and Analysis Workshop*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991.

[6] Luqi, "Software Evolution via Rapid Prototyping", *Computer*, vol. 22, pp. 13-25, 1989.

[7] P. Narendran and J. Stillman, "Formal Verification of the Sobel Image Processing Chip", *Proc. 25th ACM/IEEE Design Automation Conference*, pp. 211-217, 1988.

[8] F. Jahanian, R. Lee, and A. Mok, "Semantics of Modechart in Real Time Logic", *Proc. Hawaii International Conference of System Science*, Jan. 1988, pp. 479-489.

[9] C. Heitmeyer and B. Labaw, "Specifying Hard Real-Time Software: Experience with a Language and a Verifier", *Real-Time Systems Newsletter,* vol. 7, pp. 63-68, 1991.

[10] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison Wesley, 1991.

[11] Luqi and V. Berzins, "Rapidly Prototyping Real-Time Systems", IEEE Software, Sept. 1988, pp. 25-36.

[12] Luqi, "Real-Time Constraints in a Rapid Prototyping Language", Journal of Computer Languages, Spring, 1992.

[13] Luqi and M.T. Shing, "CAPS - A Tool for Real-Time System Development and Acquisition", *Naval Research Review*, Summer, 1992.

[14] *Ada 9x Project Report: Ada 9x Requirements*, Office of the Under-secretary of Defense for Acquisition, Washington, D.C., December 1990.

[15] Luqi, M. Shing, and V. Berzins, "Systematic Development of Hard Real-Time Software," NRL Report, 1991.

[16] *Requirements Analysis & Design Tools Report*, Software Technology Support Center, Hill AIr Force Base, UT 84056, April 1992.

[17] Luqi, "A Graph Model for Software Evolution", IEEE Transactions on Software Engineering 16(8): 917-927, Aug. 1990.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center
Cameron Station
Alexandria, VA 22314 — 2

Dudley Knox Library
Code 52
Naval Postgraduate School
Monterey, CA 93943-5100 — 2

Office of Research Administration
Code 08
Naval Postgraduate School
Monterey, CA 93943-5100 — 1

Chief of Naval Research
800 N. Quincy Street
Arlington, VA 22302-0268 — 1

Center for Naval Analysis
4401 Ford Avenue
Alexandria, VA 22302-0268 — 1

Chief of Naval Operations
Director, Information Systems (OP-945)
Navy Department
Washington, D.C. 20350-2000 — 1

Chairman, Code CS
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100 — 2

Dr. Luqi
Computer Science Department, Code CSLq
Naval Postgraduate School
Monterey, CA 93943-5100 — 10

Dr. Mantak Shing
Computer Science Department, Code Sh
Naval Postgraduate School
Monterey, CA 93943-5100 — 10

Dr. Valdis Berzins
Computer Science Department, Code CSBe
Naval Postgraduate School
Monterey, CA 93943-5100 10

Dr. Thomas Wu
Computer Science Department, Code CSWq
Naval Postgraduate School
Monterey, CA 93943-5100 1

Dr. Louis Chmura
Naval Research Laboratory
Code 8140
4555 Overlook Ave. 500 SW
Washington, DC 20375-5000 2

DUDLEY KNOX LIBRARY
3 2768 00339880 1